VISIT…

Programme

Search C#Today Living Book

Index Full Text

Advanced Search

CATEGORIES Go ▶ HOME ▶ SITE MAP ▶ SEARCH ▶ REFERENCE ▶ FORUM ▶ FEEDBACK ▶ ADVERTISE ▶ SU

The C#Today Article
August 21, 2001

Previous article - August 20, 2001

Next art August

Building an Online Shopping Cart using C# and ASP.NET Part 1

by Juan Martínez

CATEGORY: Application Development
ARTICLE TYPE: Tutorial

Reader Comments

ABSTRACT

In this article Juan Martínez discusses and implements the building blocks of an online shopping cart - covering in this part item catalogs, item details, a cart and a checkout system. This acts as the foundation for further articles, where Juan covers other functionality like administration pages and setting up credit card payments.

Article
Usefu
Innov
Inforn
15 resp

Buy this Article

Article Discussion Rate this article Related Links Index Entries

ARTICLE

Editor's Note: This article's code has been updated to work with the final release of the .Net framework.

As web developers we are required to face a wide variety of application needs. Each web site developed is unique and furnished according to each client's specific needs. The fact is that websites are indeed fabricated for each client with different specifications but every site shares some common characteristics. Those parts of the site that share functionality features can be treated as separate applications to be reused.

In this case we will address the development cycle for one of the common blocks in today's websites - an Online Shopping Cart.

We will analyze the development of a shopping cart as a group of components described clearly before we implement them, thus allowing us to use this knowledge in areas other than ASP.NET. After a description of each component the implementation will be explained using C#.

The application will be designed to work with a SQLServer database for storage. Application logic will be done within the Web Form and presented to the user through the web browser. Core logic will reside in a separate C# component using the code behind technique. It will also be .NET framework Beta 2 compliant.

It is assumed that you have regular knowledge of the C# language, web development knowledge and database design basics.

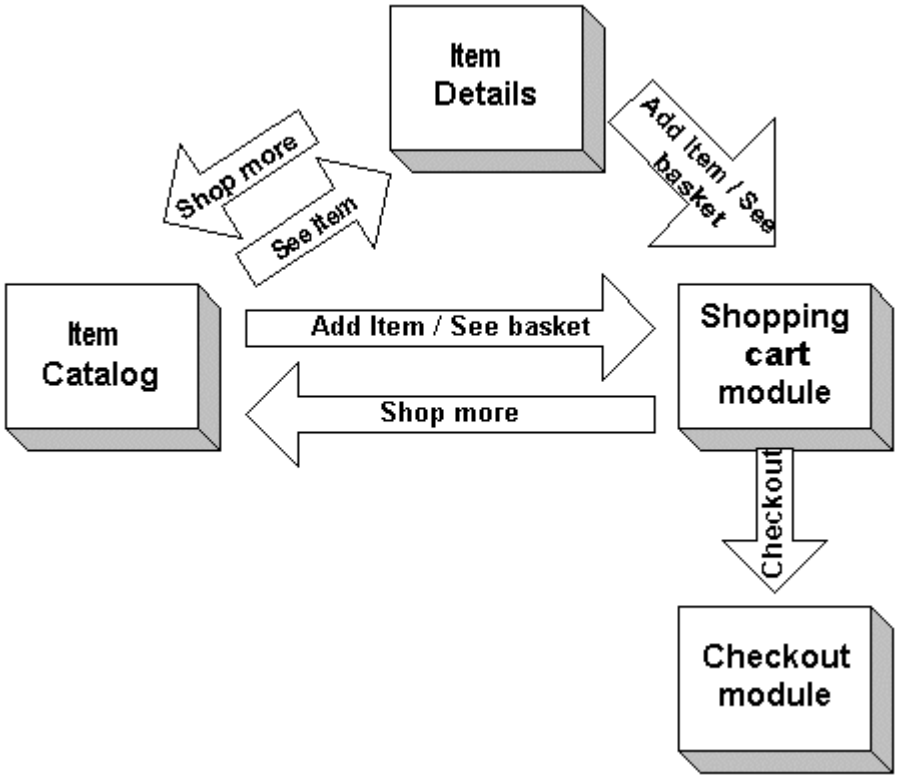
Setting up the basis

First we will take some time to understand how a simple shopping cart works. From this knowledge we will draw some conclusions and state our requirements. From these requirements the database design will emerge.

After these steps we will have a clear path of development and be ready to implement our online shopping cart in the C# language.

Digging in the Online Shopping Cart Model

We will first take a look at a simplified diagram of an Online Shopping Cart. These are the functionality blocks to be discussed.



We then have four basic modules:

- **Item Catalog** - Here we display the options to our clients in an organized way.
- **Item Details** - Here we show the client as much info as we can to show off our product.
- **Shopping Cart** - Here we manage the user's selected items.
- **Checkout System** - Here we save the items selected and the client's information to close the transaction.

These are the basic blocks to be implemented in our online store. It includes the indispensable functionality that will be described in detail later. These blocks could be enriched further with more features, which will be covered in later articles.

Defining requirements

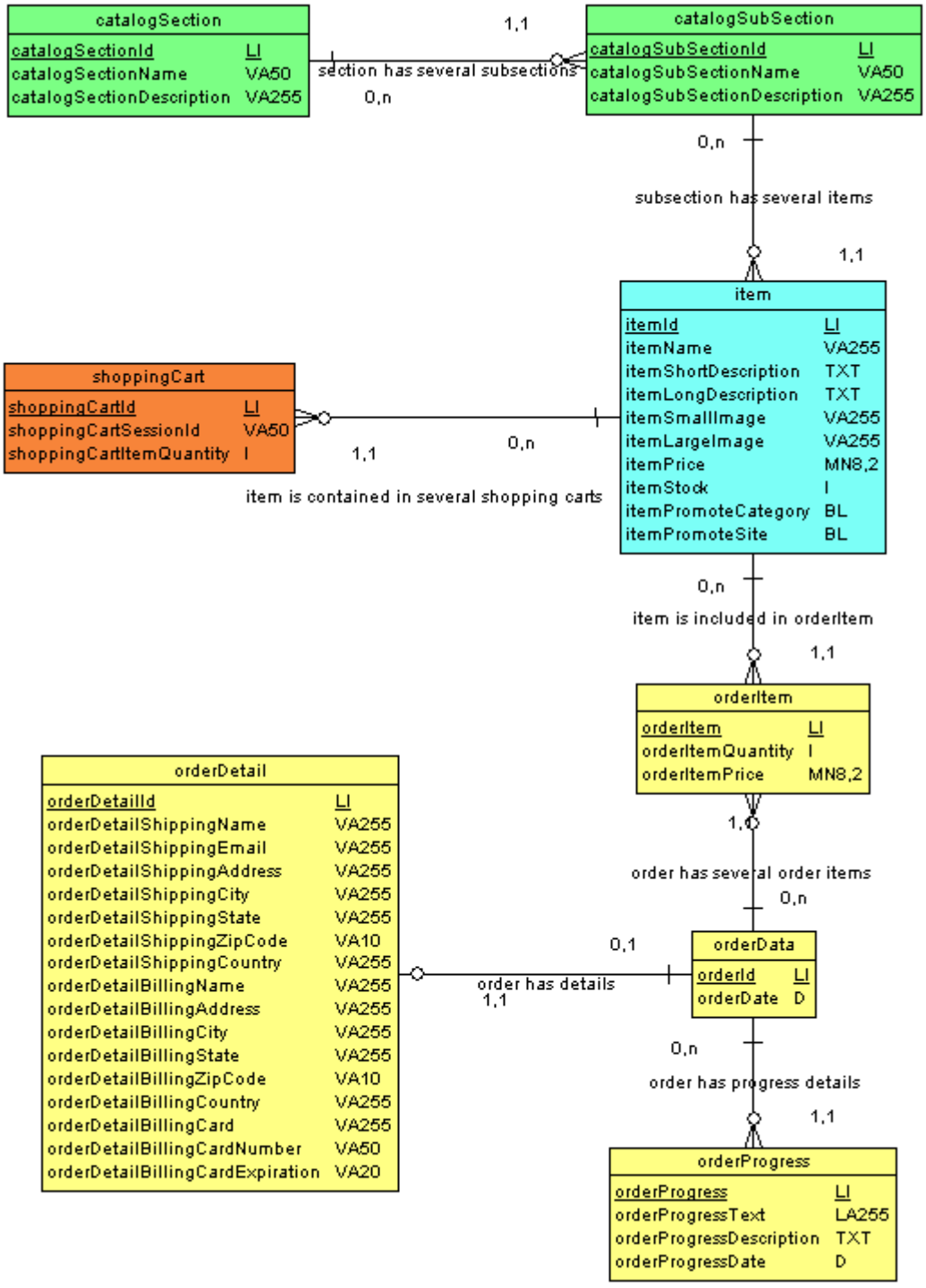
As in every software development cycle, we need to define our requirements first so we can design software capable of giving satisfaction to our customers.

Our online shopping cart application should do the following:

- Have a list of categories and subcategories.
- Items should be arranged in its corresponding subcategory.
- Items could be selected for category and home promotion.
- Each product should have an id, name, short and long descriptions, small and large images, stock and price.
- Users should be able to add products to the basket and remove them.
- The user should be given an order number and will be able to track it through an order tracking system.

Generating our database model

From our requirements we define the database schema. The tables are shown as a conceptual model, with all tables used in this version of the shopping cart.



The tables are grouped as follows:

- Green - The catalog part of our application.
- Sky blue - Item details.
- Orange - Shopping cart basket.
- Yellow - The checkout system.

Catalog Section

The green part corresponds to the **catalog** section. It is composed of two tables. This is a very simple arrangement in which the **subsection** table inherits the **section Id**. This way we can display the items in a

section/subsection approach. As this approach will work in most cases, you could improve this to a completely flexible design using recursive sections, which will inherit a parent Id.

Item Details

The item details part of our model is a trimmed down version of an item details design. We have only one table in which we save the vital information of the **item** such as description, price and images. We have a couple of Boolean values that are used to specify if the item should be displayed as a home or section item.

Shopping cart basket

The shopping basket is a simple table that is used temporarily to store the items selected by the user. We just save the **session Id**, the **item id** and the **quantity**. We will implement this as an **in-memory data table** and hold our selected items in a session variable. This table is shown to show you're the information that needs to be retained during our session.

Checkout system

This is the most complicated part or our system. Upon checkout we create an order Id in the order table. Then the items stored in the shopping basket are transferred and saved in our order Items table. The user information is stored in the order details table. These two tables inherit the order Id. We have a fourth table which also inherits the order Id. This is the order Progress table and it is used for order tracking. As progress is done, the online shop administrator should add a record to this table indicating the progress done to date. This is then checked by the buyer.

Let the coding begin

As our design basis is done, we are all set to start our coding. The architecture will be based on simple Web Forms calling custom User Controls. Core logic will be done in separate components using the code behind technique.

Application framework

We start by setting our basic framework. We make use of our `web.config` file to save important application information such as our connection string. This is how our configuration file will look:

```
<configuration>
  <appSettings>
    <add key="connString" value="server=(local)\NetSDK;database=ShoppingCart;Trusted_Con
  </appSettings>
</configuration>
```

To retrieve this information we use the following code snippet.

```
String connString = ConfigurationSettings.AppSettings["connString"];
```

As you can see, this is a very easy and convenient way to store application wise data.

Category List Block

All being set, we have our first task which is to display a list of products which will be filtered by section and subsection if selected.

The first task is to set up the workspace of our Web Forms. We set up our language and add the appropriate namespaces for our code to work.. We set C# as our language and import the `System` namespace for general purposes. We then add the `System.Data` and `System.Data.OleDb` since we will be using SQLServer as our database engine. We should set the debug flag to false once we deliver this application to the real world.

```
<%@ Page Language="C#" Debug="true" %>
<%@ Import Namespace="System" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient " %>
```

We now set our references to our custom **User Controls**.

```
<%@ Register TagPrefix="SC" TagName="Site" Src="uc_header.ascx" %>
<%@ Register TagPrefix="SC" TagName="Section" Src="uc_catalog_section.ascx" %>
<%@ Register TagPrefix="SC" TagName="SubSection" Src="uc_catalog_subsection.ascx" %>
<%@ Register TagPrefix="SC" TagName="ProductList" Src="uc_catalog_product_list.ascx" %>
```

We reference our controls by embedding them into the aspx file.

```
<SC:Site runat="server" />
```

This will be our complete catalog WebForm, the `catalog.aspx` file.

```
<%@ Page Language="C#" Debug="true" %>
<%@ Register TagPrefix="SC" TagName="Site" Src="uc_header.ascx" %>
<%@ Register TagPrefix="SC" TagName="Section" Src="uc_catalog_section.ascx" %>
<%@ Register TagPrefix="SC" TagName="SubSection" Src="uc_catalog_subsection.ascx" %>
<%@ Register TagPrefix="SC" TagName="ProductList" Src="uc_catalog_product_list.ascx" %>
<html>
<head>
<title>Shopping Cart in C# - Catalog</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
<form runat=server>
<SC:Site runat="server" />
<table width="800" cellspacing="0" cellpadding="3" border="0" style="border-color:Black;
<tr>
<td width="800" align="right" bgcolor="#CCCCCC"><span class="itemTitle">Item Catalog</sp
</tr>
</table>
<table width="800" cellspacing="0" cellpadding="0" border="0">
<tr>
<td width="200" height="100%" rowspan="2" valign="top"><SC:Section runat="server" /></td
<td width="600" height="10" valign="top"><SC:SubSection runat="server" /></td>
</tr>
<tr>
<td width="600" height="100%"><SC:ProductList runat="server" /></td>
</tr>
</table>
</form>
</body>
</html>
```

We have used four custom user controls in our first Web Form. We shall now go towards implementing these custom controls, most of which, will be reused through out the site.

Lets start by implementing the simplest of our user controls, `uc_header.ascx`. This file contains the "logo" of the site and a couple of links. We encapsulate this into a control since it will be repeated through the whole site. This is the code:

```
<table width="800" cellspacing="0" cellpadding="3" border="0" style="border-color:Black;
<tr>
<td width="400"><a href="catalog.aspx"><span class="mainTitle">Shopping Cart Site</span>
<td width="400" align="right"><a href="cart.aspx">View Cart</a><span class="mainText">
| </span><a href="tracking.aspx">Track your order</a></td>
</tr>
</table>
```

As you can see this is just a static table, reminiscent of old style include files. Next we code a more complex user control, `uc_catalog_section.ascx` which displays the list of sections available on the site. This is done by binding a data retrieval codebehind class to the user control.

This is our User Control Code:

```
<%@ Control Language="C#" Debug="true" Inherits="CodeBehind.UcSection" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
```

```
<html>
<head>
   <title></title>
    <script language="C#" runat="server"></script>
</head>
<body>

    <font face="Verdana" size="-1">
        <asp:DataList id="MySectionList" runat="server"
            BorderColor="black"
            BorderWidth="1"
            GridLines="Both"
            CellPadding="3"
            Font-Name="Verdana"
            Font-Size="8pt"
            Width="200px"
            HeaderStyle-BackColor="#aaaadd"
            SelectedItemStyle-BackColor="Gainsboro"
            >
              <HeaderTemplate>
                Sections
              </HeaderTemplate>
              <ItemTemplate>
              <a href="catalog.aspx?sectionId=<%# DataBinder.Eval(Container.DataItem,"ca
                                                                  &sectionIndex=<%#Cont
                                        <%# DataBinder.Eval(Container.DataItem, "catalogS
              </ItemTemplate>
              <SelectedItemTemplate>
                <%# DataBinder.Eval(Container.DataItem, "catalogSectionName") %>
              </SelectedItemTemplate>
        </asp:DataList>
    </font>

</body>
</html>
```

We place some more html into the control and we drop a DataList object to handle the rendering of the information. The `<%# DataBinder.Eval(Container.DataItem, "catalogSectionName") %>` code is used to select specific information from the bound data and put it in place.

In our codebehind class we first have to declare our DataList as protected. We then do some database work. Once we have the information we need ready from our database, we bind this data to our Web Control, the DataList.

Our User Control inherits the code behind class and that way they can work together. It is important to declare your shared variables (such as Web Form Controls) as protected so that the can be reached within the code behind class.

```
public class UcSection : UserControl {
  protected DataList MySectionList;
  protected Label MyLabel;

  protected void Page_Load(object sender, EventArgs e) {
    if (!IsPostBack) {
      Bind_MySectionList();
    }

    if (Request.QueryString["sectionIndex"] != null) {
      MySectionList.SelectedIndex = Int32.Parse(Request.QueryString["sectionIndex"]);
    }
  }

  protected void Bind_MySectionList() {
    String connString = ConfigurationSettings.AppSettings["connString"];
    SqlConnection myConnection = new SqlConnection(connString);
    SqlDataAdapter myCommand = new SqlDataAdapter("SELECT catalogSectionId, catalogSecti
                               FROM catalogSection ORDER BY catalogSectionName", myConnec
    DataSet ds = new DataSet();
    myCommand.Fill(ds, "catalogSection");
```

```
    MySectionList.DataSource = new DataView(ds.Tables[0]);
    MySectionList.DataBind();
  }
}
```

We then have to implement our **subsection user control**. This control will do a similar task to the section control. We will just add a filter to the Query applied to the database. We will select subsections corresponding to the selected section. The rest is pretty much the same.

```
"SELECT catalogSubSectionId, catalogSubSectionName FROM catalogSubSection WHERE catalogS
                                                        sectionId + " ORDER BY catalogSu
```

We then need to display our product list. We have three types of products:

- Site Products: These are shown if no section is selected
- Section Products: These are shown if a section is selected but no subsection is selected.
- Normal Products: These are products that belong to the selected subsection.

We need to filter the three possibilities and create the proper query. This is done in the `UcProductList` class with this code. After we have our query we bind it to a DataGrid Web Control to display the items in a table arrangement.

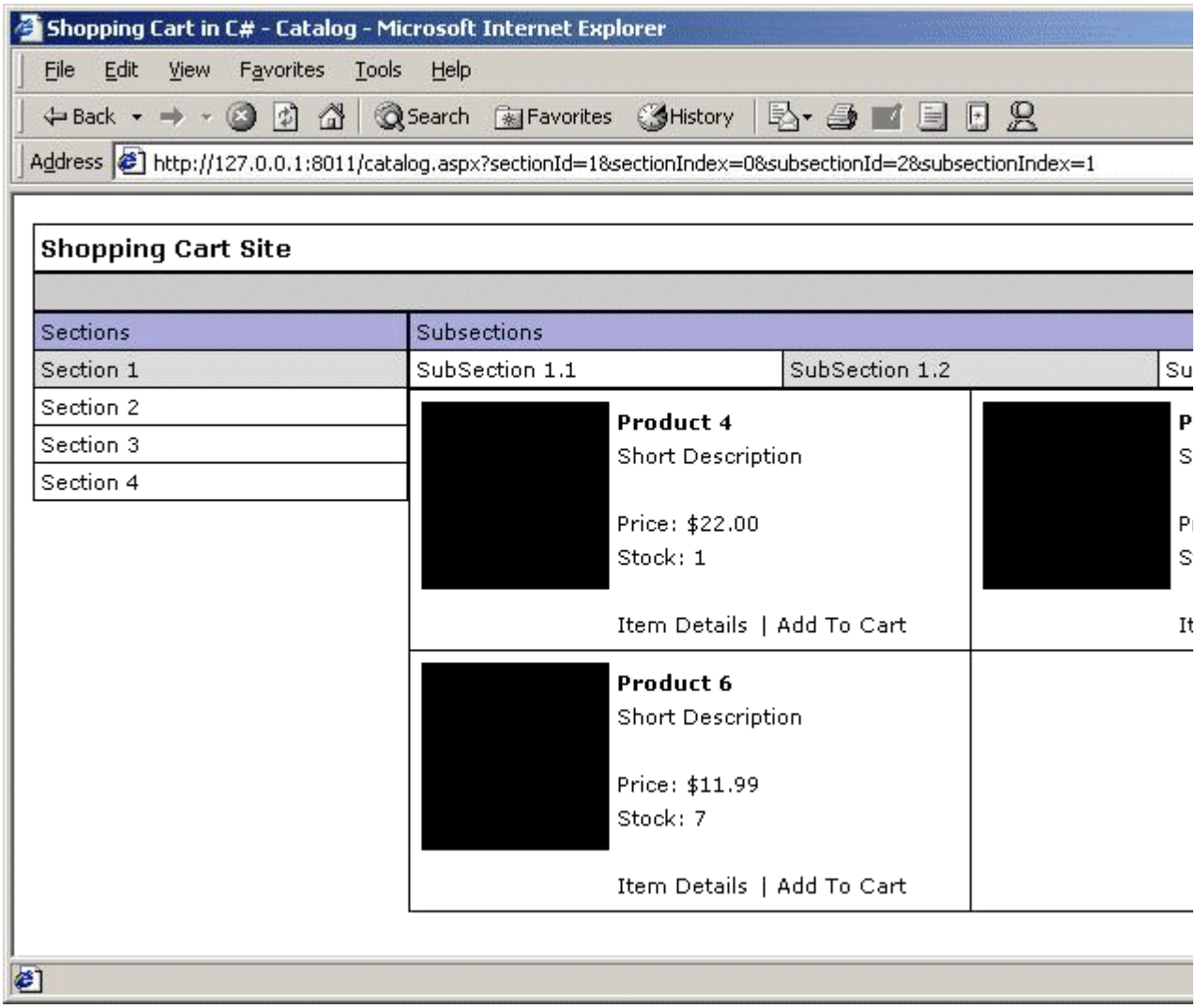
```
string SQLQuery = "SELECT itemId, itemName, itemShortDescription, itemSmallImage, itemPr
                                                  FROM item WHERE itemPromoteSite=1 ORDER
int mysectionId = 0;
if (Request.QueryString["sectionId"] != null) {
  mysectionId = Int32.Parse(Request.QueryString["sectionId"]);
}
if (mysectionId != 0) {
  //If we have a section selected we filter products for this section
  SQLQuery = "SELECT item.itemId, item.itemName, item.itemShortDescription, item.itemSma
     FROM (catalogSection INNER JOIN catalogSubSection ON catalogSection.catalogSectionI
                                        INNER JOIN item ON catalogSubSection.catalog
                          WHERE (((catalogSection.catalogSectionId)=" + mysectionId + "
}

int mysubsectionId = 0;
if (Request.QueryString["subsectionId"] != null) {
  mysubsectionId = Int32.Parse(Request.QueryString["subsectionId"]);
}
if (mysubsectionId != 0) {
  //If we have a subsection selected we filter products for this subsection
  SQLQuery = "SELECT itemId, itemName, itemShortDescription, itemSmallImage, itemPrice,
  WHERE catalogSubSectionId=" + mysubsectionId + " ORDER BY itemName";
}
```

This concludes our catalog presentation layer which consists of four main user controls.

- The first control is responsible for displaying a list of sections.
- The second control is responsible for the subsection list.
- The third control is responsible for displaying a grid of items. This list of items is retrieved from one of three sql queries depending on the situation.
- A final control renders the top of the page.

All the controls work together to form `catalog.aspx`. The user controls make use of codebehind classes for database access and bind the DataLists in our page.

Our `catalog.aspx` file should yield something like this:



Item Details

Our next step in building our shopping cart is to show off the item details. We first will use some of the code done for the catalog presentation before. We will take the basic html framework and three user controls. We will reuse the header, section and subsection control.

The difference here will be to replace the **item list** control with a **new item details** control. We will use a data list Web Control to display the items characteristics as well as its corresponding class in our code behind repository.

The code to do this simple task is as follows for the user control:

```
<%@ Control Language="C#" Debug="true" Inherits="CodeBehind.UcItemDetails" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<script language="C#" runat="server"></script>
<font face="Verdana" size="-1">
<asp:DataList id="MyItemDetails" runat="server"
  BorderColor="black"
  BorderWidth="1"
  GridLines="Both"
  CellPadding="3"
  Font-Name="Verdana"
  Font-Size="8pt"
  Width="600px"
  HeaderStyle-BackColor="#aaaadd"
  SelectedItemStyle-BackColor="Gainsboro"
  RepeatDirection = "Horizontal"
  RepeatColumns = "1"
```

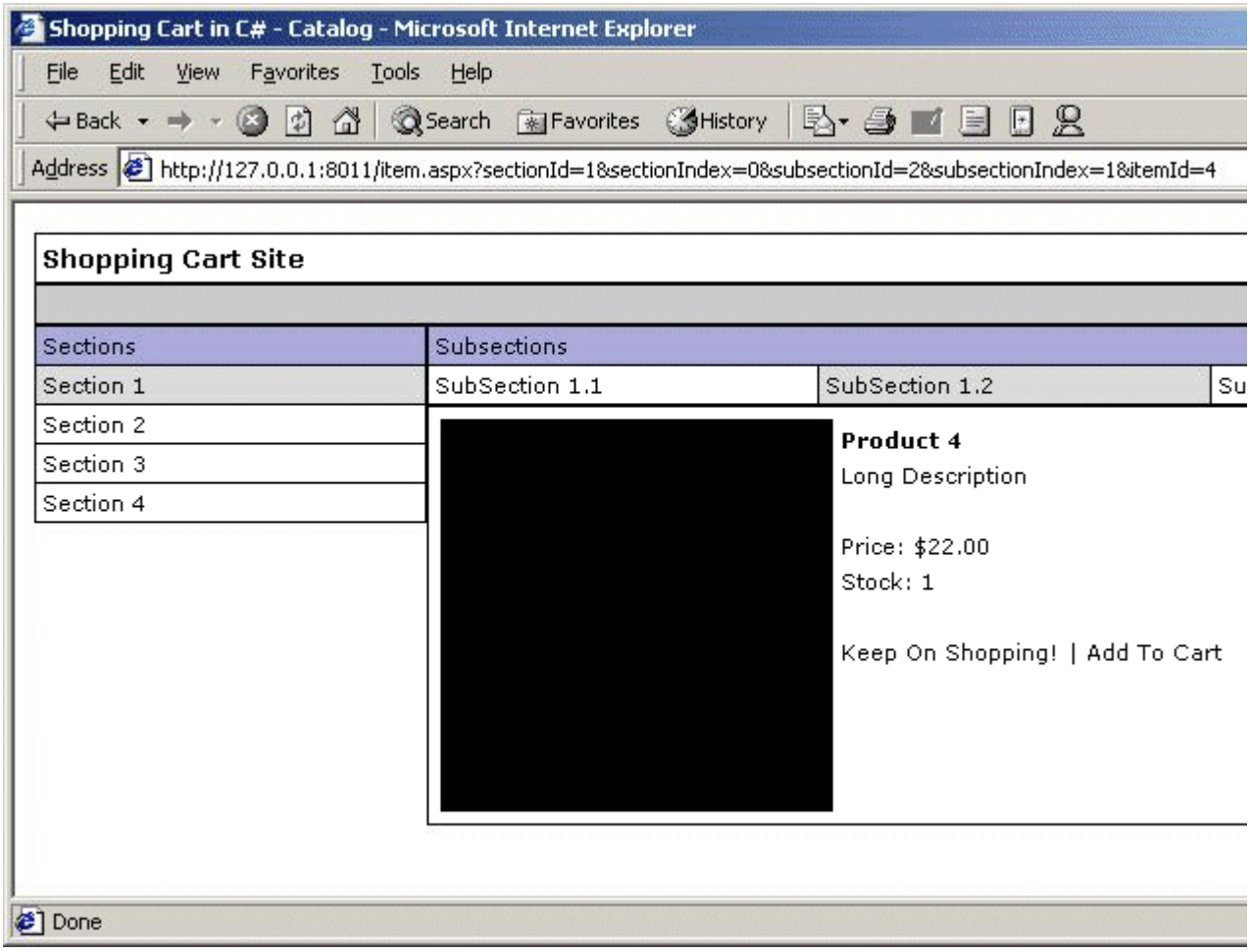
```
  >
  <ItemTemplate>
    <table>
    <tr>
    <td valign="top">
      <img src="images/<%# DataBinder.Eval(Container.DataItem, "itemLargeImage") %>" wid
    </td>
    <td valign="top">
      <span class="itemTitle"><%# DataBinder.Eval(Container.DataItem, "itemName") %></sp
      <br>
      <span class="itemText"><%# DataBinder.Eval(Container.DataItem, "itemLongDescriptio
      <br><br>
      <span class="itemText">Price: <%# (DataBinder.Eval(Container.DataItem, "itemPrice"
      <br>
      <span class="itemText">Stock: <%# DataBinder.Eval(Container.DataItem, "itemStock")
      <br><br>
      <a href="catalog.aspx?sectionId=<%=sectionId%>&sectionIndex=<%=sectionIndex%>&subs
                                                    &subsectionIndex=<%=subsectionInde
      <span class="itemText"> | </span>
      <a href="cart.aspx?cartAction=1&itemId=<%# DataBinder.Eval(Container.DataItem, "it
    </td>
    </tr>
    </table>
  </ItemTemplate>
  </asp:DataList>
</font>
```

And our code behind class:

```
protected void Bind_ItemDetails() {
  int myItemId = 0;
  if (Request.QueryString["itemId"] != null) {
    myItemId = Int32.Parse(Request.QueryString["itemId"]);
  }
  string SQLQuery = "SELECT itemId, itemName, itemLongDescription, itemLargeImage, itemP
                                                          FROM item WHERE itemId
  String connString = ConfigurationSettings.AppSettings["connString"];
  SqlConnection myConnection = new SqlConnection(connString);
  SqlDataAdapter myCommand = new SqlDataAdapter(SQLQuery, myConnection);
  DataSet ds = new DataSet();
  myCommand.Fill(ds, "item");

  MyItemDetails.DataSource = new DataView(ds.Tables[0]);
  MyItemDetails.DataBind();
}
```

Further enhancements can be done to this simple module. This form shows the stored details of the item and can be enriched with user reviews and related item lists.

Our resulting screen should look as follows:



Shopping cart basket

The shopping cart basket implements the majority of the functionality. It should provide ways to add, edit and delete items from it. The basket is merely a temporal storage area for the user to group the items of interest. We then need some session management to be able to recognize which items belong to each client in its own session.

ASP.NET, as in previous asp applications, provides the necessary objects to work with sessions. In this case we will take advantage of the **Session** object. We will store a **DataTable** object as a Session variable therefore all shopping cart basket operations are done in memory. This is a very fast approach but will need to be provided with enough memory to handle concurrent clients.

The shopping basket module comprises two fundamental parts. The first involves managing the items that the user wants to have in the basket. The second is in charge of displaying the items currently in the basket. We will see the details now.

We instruct our Web Form to add an item to the cart with querystring variables. We just add an item to the DataTable. To implement the update and delete methods we bind the DataTable to the DataList Web Control and perform the operations.

```
void Page_Load(Object Sender, EventArgs E) {
   if (Request.QueryString["cartAction"] != null) {
      this.cartAction = Int32.Parse(Request.QueryString["cartAction"]);
   }

   if (Session["ShoppingCart"] == null) {
      Cart = new DataTable();
      Cart.Columns.Add(new DataColumn("ItemId", typeof(string)));
      Cart.Columns.Add(new DataColumn("Item", typeof(string)));
      Cart.Columns.Add(new DataColumn("Qty", typeof(string)));
      Cart.Columns.Add(new DataColumn("Price", typeof(string)));
      Session["ShoppingCart"] = Cart;
   }
   else {
```

```
      Cart = (DataTable)Session["ShoppingCart"];
   }

   CartView = new DataView(Cart);
   CartView.Sort = "Item";

   if (!IsPostBack) {
      //Add new entry to the shopping cart
      int myCartAction = 0;
      if (Request.QueryString["cartAction"] != null) {
         myCartAction = Int32.Parse(Request.QueryString["cartAction"]);
      }
      if (myCartAction==1) {
         //Take the items details from the database
         string myItemId = "0";
         if (Request.QueryString["itemId"] != null) {
            myItemId = Request.QueryString["itemId"];
         }
         string SQLQuery = "SELECT itemName, itemPrice FROM item WHERE itemId = " + myIt
            String connString = ConfigurationSettings.AppSettings["connString"];

              SqlConnection myConnection = new SqlConnection(connString);
                 SqlCommand myCommand = new SqlCommand(SQLQuery, myConnection);
             myConnection.Open();
            SqlDataReader dr2 = myCommand.ExecuteReader();

         if (dr2.Read()) {
                DataRow dr = Cart.NewRow();
              dr[0] = myItemId;
              dr[1] = dr2.GetString(0);
              dr[2] = "1";
            dr[3] = dr2.GetSqlMoney(1).ToString();
             Cart.Rows.Add(dr);
         }
      }

      BindCartList();
      BindTotalList();
   }
}
```

The functions to update and delete data are as follows:

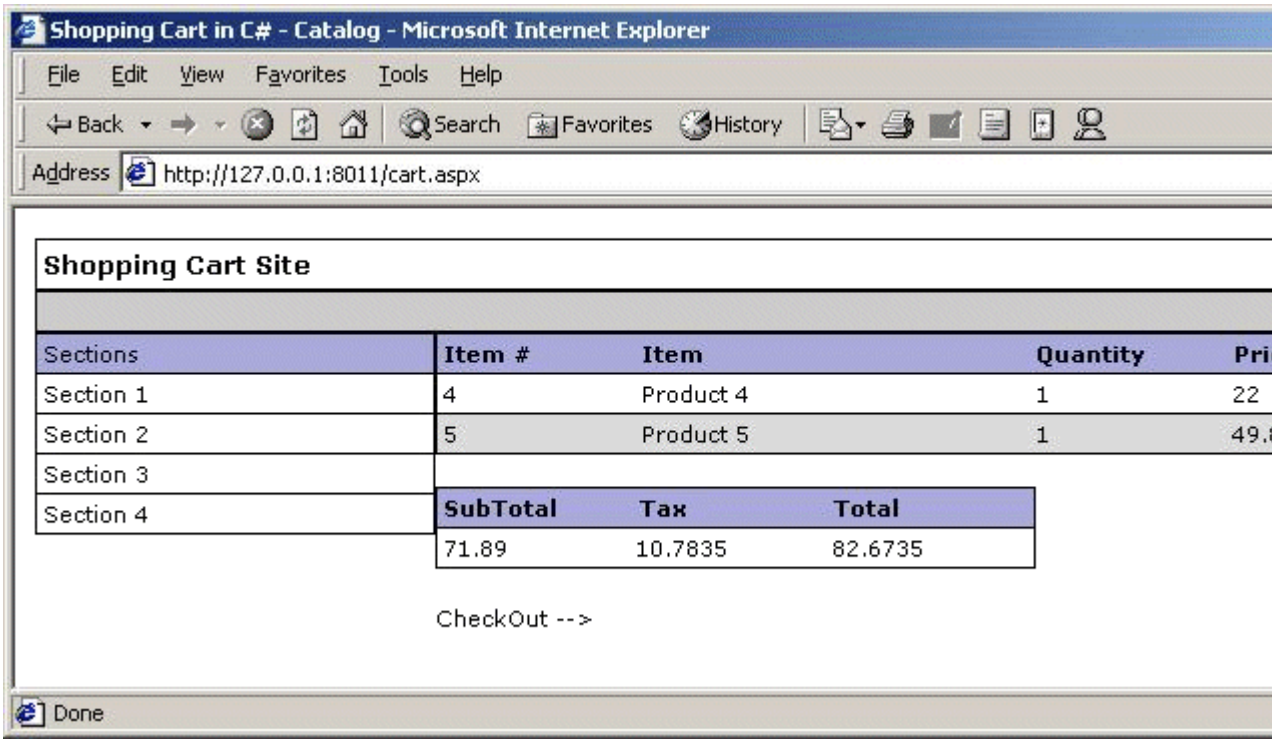
```
    protected void DataCartList_DeleteCommand1(Object Sender, DataListCommandEventArgs e
        string item = ((Label)e.Item.FindControl("Label4")).Text;
   CartView.RowFilter = "Item='"+item+"'";
        if (CartView.Count > 0) //item exists in cart
            CartView.Delete(0);
        CartView.RowFilter = "";

        CartList.EditItemIndex = -1;
        BindCartList();
   BindTotalList();
    }

    protected void DataCartList_EditCommand1(Object Sender, DataListCommandEventArgs e)
        CartList.EditItemIndex = (int)e.Item.ItemIndex;
        BindCartList();
   BindTotalList();
    }

    protected void DataCartList_CancelCommand1(Object Sender, DataListCommandEventArgs e
        CartList.EditItemIndex = -1;
        BindCartList();
   BindTotalList();
    }

    protected void DataCartList_UpdateCommand1(Object Sender, DataListCommandEventArgs e
   string itemId = ((Label)e.Item.FindControl("Label1")).Text;
        string item = ((Label)e.Item.FindControl("Label2")).Text;
```

```
        string qty = ((TextBox)e.Item.FindControl("Text1")).Text;
        string price = ((Label)e.Item.FindControl("Label3")).Text;

        // with a database, we'd use an update command.  Since we're using an in-memory
        // DataTable, we'll delete the old row and replace it with a new one
        //remove old entry
        CartView.RowFilter = "Item='"+item+"'";
        if (CartView.Count > 0) //item exists in cart
            CartView.Delete(0);
        CartView.RowFilter = "";

        //add new entry
        DataRow dr = Cart.NewRow();
        dr[0] = itemId;
        dr[1] = item;
        dr[2] = qty;
   dr[3] = price;
        Cart.Rows.Add(dr);

        CartList.EditItemIndex = -1;
        BindCartList();
   BindTotalList();
    }
```

Finally we need to calculate the subtotal, tax and total cost of the order. We use the `BindTotalList()` function to generate content and bind it to another Web Control in our page.

```
protected void BindTotalList() {
   double mySubTotal = 0;
   double myTaxRate = 0.15;
   double myTax = 0;
   double myTotal = 0;

   DataTable CartTable = (DataTable)Session["ShoppingCart"];
   foreach(DataRow myRow in CartTable.Rows) {
      Double tempPrice = Double.Parse(myRow[3].ToString());
      Int32 tempQty = Int32.Parse(myRow[2].ToString());
      mySubTotal += tempPrice * tempQty;
   }

   myTax = mySubTotal * myTaxRate;
   myTotal = mySubTotal + myTax;

   CartTotal = new DataTable();
       CartTotal.Columns.Add(new DataColumn("SubTotal", typeof(string)));
       CartTotal.Columns.Add(new DataColumn("Tax", typeof(string)));
       CartTotal.Columns.Add(new DataColumn("Total", typeof(string)));

   DataRow drTotal = CartTotal.NewRow();
       drTotal[0] = mySubTotal.ToString();
       drTotal[1] = myTax.ToString();
       drTotal[2] = myTotal.ToString();
   CartTotal.Rows.Add(drTotal);

   CartTotalView = new DataView(CartTotal);
   CartTotalList.DataSource = CartTotalView;
   CartTotalList.DataBind();
}
```

The shopping cart screen looks like this:



Checkout System

The checkout system has the responsibility of saving the client info and selected items for processing. We use three tables to save the client info. These are the following:

- orderData - Stores the date of the order and generates the order Id.
- orderItem - List of items bought, it also saves the price at the time of purchase.
- orderDetail - Saves shipping and billing information for the order.

We divide our process into three steps:

- Shipping and Billing Information - Here we get the shipping information. With this info we can calculate the shipping costs. This should be done accordingly to each store's necessities. We validate credit card information also. This should be done using a specific provider like paypal.
- Confirmation - Here we present a summary of the information received and show the list of items to be bought, as well as the grand total of the order. The user is asked to submit the information if it is correct.
- Summary - We finish the transaction and show the user the order number.

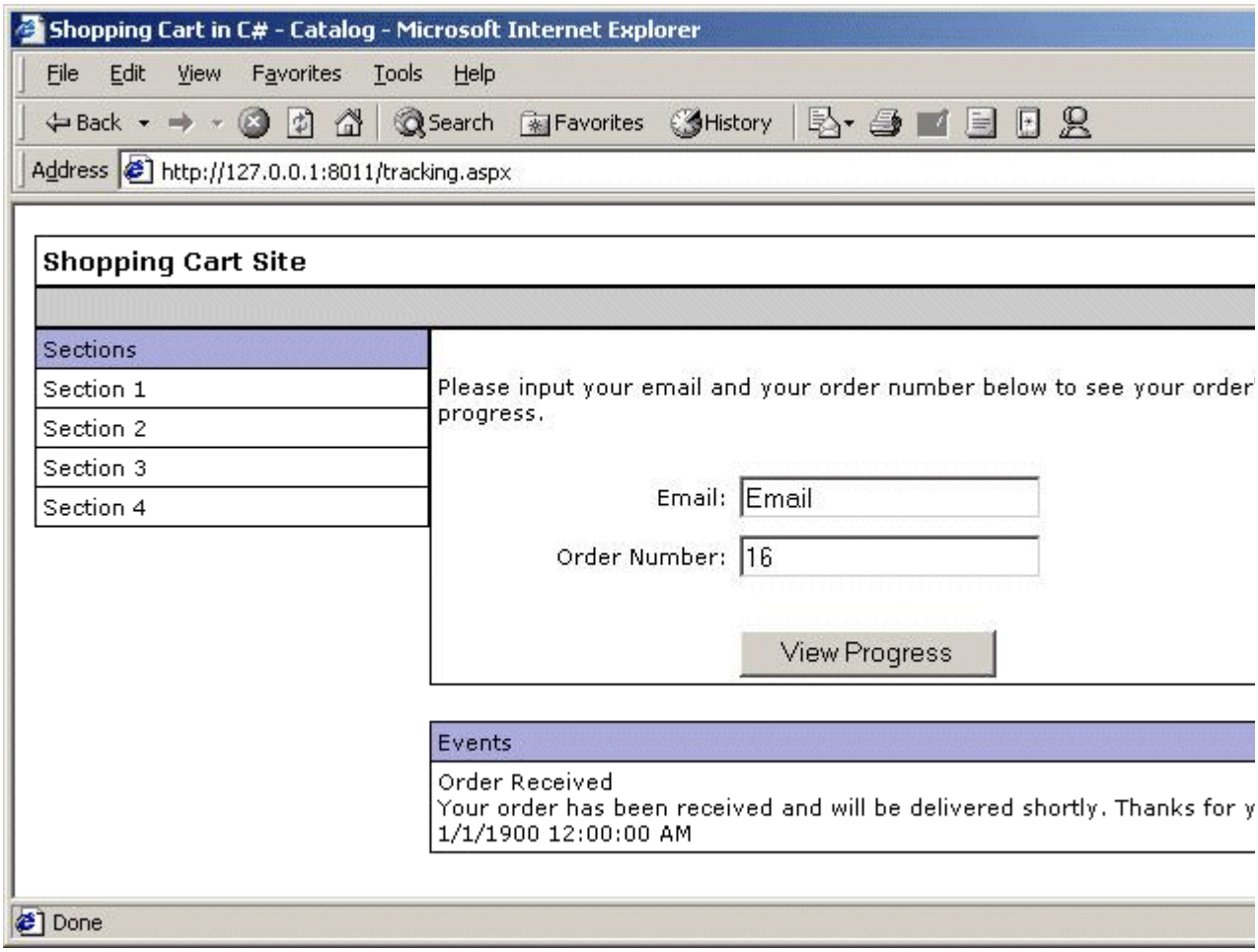
We post the info from the first to the second step. We then use Web Form Controls to ask for confirmation. If the info is correct, then we register the order for our client.

We do need to implement some form validation before posting the information and implement some credit card validation through an external provider or your own software.

Tracking your order

The final part of our online shopping cart application is the tracking system. We will implement a basic tracking system which will ask the user for their email and order number. The system will then show a list of milestones registered in the tracking database.

We use a single table to save the tracking information. It consists of the order id and the information of the goals achieved. Upon valid email and order number we display the list of events below.



This is the code required:

```
protected void ButtonView_Click(object Source, EventArgs e) {
   //Look for order id with provided email and order number
   String connString = ConfigurationSettings.AppSettings["connString"];
   string strSelect = "SELECT orderId FROM orderDetail WHERE orderId=" + orderIdBox.Text
   AND orderDetailShippingEmail='" + emailBox.Text + "'";
   SqlConnection myConnection = new SqlConnection(connString);
   SqlCommand myCommand = new SqlCommand(strSelect, myConnection);

   myConnection.Open();
   SqlDataReader dr = myCommand.ExecuteReader();
   int newOrderId = 0;
   if (dr.Read()) {
      //emailBox.Text = "Encontramos!!!";
      Bind_TrackingList();
   }
   myConnection.Close();
}

protected void Bind_TrackingList() {
   string SQLQuery = "SELECT orderProgressText, orderProgressDescription, orderProgressD
   WHERE orderId = " + orderIdBox.Text;
   String connString = ConfigurationSettings.AppSettings["connString"];

   SqlConnection myConnection = new SqlConnection(connString);
   SqlDataAdapter myCommand = new SqlDataAdapter(SQLQuery, myConnection);

   DataSet ds = new DataSet();
   myCommand.Fill(ds, "orderProgress");

   MyTrackingList.DataSource = new DataView(ds.Tables[0]);
   MyTrackingList.DataBind();
}
```

Deployment of the site

Deployment of this particular application is very simple.

- Create a **new database** in SQLServer
- Once we have created the database we need to run the **database creation script** provided.
- Create a **new site** in your IIS administration program
- Copy the **web.config** file to the wwwroot folder.
- Copy the **Web Forms** and **User controls** to the wwwroot folder.
- Copy the **codebehind.dll** into your wwwroot/bin directory (no registration required!).

Improvements to our online shopping cart application

The shopping cart we just made takes a basic approach to the application providing the central framework of an online shopping cart. Further enhancements can be made and will be the responsibility of the reader. These enhancements should be made to enrich the shopping cart application give a better service to the online community. Some ideas are presented here.

Feature improvements

- Recursive sections for n levels of profundity
- Item reviews
- Item related items
- User accounts and management
- Mailing list to customers
- Specific shipping costs calculations

Technical improvements

- Build based on components for performance and possible code reuse
- Migrate to SQL Server database using stored procedures for performance
- Implement SSL in your server (a must but out of scope!)
- Implement real time credit card processing and charging.

Conclusions

The system presented shows the basis of a working online shopping cart. It involves the main aspects of an online store which are the catalog of items, the item detail module, the shopping cart basket and the checkout module. Every shopping cart online today must implement these basic parts to be functional. Hopefully this article has given you the knowledge to build your own with ease and to add new functionality to meet your needs as ideas for improvement have been given.

In future articles, we will cover some of the important requirements of a real - world shopping cart that have been left out of this article for the sake of brevity - a management console to add, modify and delete categories, items and orders; item reviews; SSL for credit card payments; and a discussion of working with a credit card processing provider.

RATE THIS ARTICLE

Please rate this article (1-5). Was this article...

Useful? **No** ○ ○ ○ ○ ○ **Yes, Very**

Innovative? **No** ○ ○ ○ ○ ○ **Yes, Very**

Informative? **No** ○ ○ ○ ○ ○ **Yes, Very**

Brief Reader Comments?

Your Name:

USEFUL LINKS

Related Tasks:

- **Download the support material** for this
- **Enter Technical Discussion** on this Artic
- Technical Support on this article - support(
- See other articles in the Application Develc
- See other Tutorial articles
- Reader Comments on this article
- Go to Previous Article
- Go to Next Article

(Optional)

Submit

Related Sources

- asp101: http://www.asp101.com/samples/shopping.asp
- Payment Online: http://www.paymentonline.com/
- Power Designer: http://www.sybase.com/products/enterprisemodeling/powerdesigner
- Pay Pal: www.paypal.com

Index Entries in this Article

- C#
- checkout system
- code behind technique
- contents, displaying
- contents, updating
- creating
- database design
- DataGrid
- DataList control
- DataTable object
- designing
- item catalog
- item detai
- online sto
- persistenc
- session va
- sessions
- shopping
- SQL Serve
- tracking s
- user contr
- web contr
- web forms
- web.confi

Search the C#Today Living Book

Index Full Text Advanced Search

C#Today is brought to you by Wrox Press (www.wrox.com). Please see our terms and conditions and privac
C#Today is optimised for Microsoft Internet Explorer 5 browsers.
Please report any website problems to webmaster@csharptoday.com. Copyright © 2002 Wrox Press. All Rights